---

**Article:**
Dziurzanski, Piotr, Zhao, Shuai, Przewozniczek, Michael et al. (2 more authors) (2020) Scalable Distributed Evolutionary Algorithm Orchestration using Docker Containers. Journal of Computational Science. 101069. pp. 1-14. ISSN 1877-7503

https://doi.org/10.1016/j.jocs.2019.101069

---

# Scalable Distributed Evolutionary Algorithm Orchestration using Docker Containers

Piotr Dziurzanski[a], Shuai Zhao[a,*], Michal Przewozniczek[a], Marcin Komarnicki[b], Leandro Soares Indrusiak[a]

[a] *Department of Computer Science, University of York, Deramore Ln, Heslington YO10 5GH, York, UK*
[b] *Department of Computational Intelligence, Wroclaw University of Technology, Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland*

**Abstract**

In smart factories, integrated optimisation of manufacturing process planning and scheduling leads to better results than a traditional sequential approach but is computationally more expensive and thus difficult to be applied to real-world manufacturing scenarios. In this paper, a working approach for cloud-based distributed optimisation for process planning and scheduling is presented. Three managers dynamically governing the creation and deletion of subpopulations (islands) evolved by a multi-objective genetic algorithm are proposed, compared and contrasted. A number of test cases based on two real-world manufacturing scenarios are used to show the applicability of the proposed solution.

*Keywords:* Smart factory; Industry 4.0; Evolutionary algorithms; Distributed optimisation; Multi-objective optimisation; Integrated process planning and scheduling

## 1. Introduction

The versatile benefits stemming from cloud computing have been widely adopted in smart factories under the name of cloud manufacturing [12]. Cloud manufacturing offers assorted services, supporting collaboration, sharing and management of manufacturing resources [33]. It can be treated as a solution to the new manufacturing challenges related to manufacturing small batches of highly customised commodities [35]. A smart factory can receive a new manufacturing order at any time and for each such order, a process planning and scheduling need to be performed with no delay [6]. Similarly, process planning and scheduling are required to be re-executed in case any unexpected event occurs in the factory,

*Corresponding author
*Email addresses:* piotr.dziurzanski@york.ac.uk (Piotr Dziurzanski), shuai.zhao@york.ac.uk (Shuai Zhao), michal.przewozniczek@york.ac.uk (Michal Przewozniczek), marcin.komarnicki@pwr.edu.pl (Marcin Komarnicki), leandro.indrusiak@york.ac.uk (Leandro Soares Indrusiak)

for example, a failure of a smart device (thing) is detected [34]. Process planning and scheduling are notorious for being computationally demanding [32] and hence require significant computational power to be performed in a time frame acceptable for a company, which may be a few orders of magnitude lower than the manufacturing process itself [6]. Between subsequent executions of the process planning and scheduling, the computational power is not needed. Consequently, the workload related to the process planning and scheduling follows the on-and-off workload pattern and, as such, satisfies two out of the three criteria for suitability for public cloud provided in [9], namely it has an unpredictable load or a potential for an explosive growth and is characterised with different computational power requirements at different time intervals. To fulfill the third criterion, the workload needs to be horizontally scalable, so the process planning and scheduling shall be parallelised easily.

Both process planning and scheduling are multi-criteria decision making problems. In a process planning, methods to manufacture a commodity are determined whereas in a project scheduling, specific tasks are assigned to specific machines for certain time slots [26]. Both these stages can be integrated into one process solvable with a typical metaheuristics, such as Genetic Algorithms (GAs) [36], usually leading to better results than a sequential execution of these two stages [26]. GA is an algorithm inspired by the process of natural selection that is often used to solve optimisation problems by applying bio-inspired operators named mutation, crossover and selection to a population of candidate solutions to generate a new population with a higher quality. Such process of generating new populations is continued until an assumed ending condition is satisfied. GAs are known to be easily parallelisable at three levels: fitness evaluation, population and individual [4]. Among these levels, the population-level parallelism (aka the Island Model) is applied in this paper. In this model, a set of populations are evolved independently but some individuals are migrated between these populations. Such island communication pattern can be significantly more advantageous than evolving a single large population [21]. Moreover, if the process planning and scheduling are performed using the Island Model of GA, the third workload criterion for cloud suitability from [9] is satisfied as well.

As the number of GA's islands can be determined dynamically based on the optimisation state [16, 27], it is important to have short island initialisation time. However, provisioning of a single virtual machine (VM) in a cloud was reported to take more than a minute [23]. Provisioning of VMs in a number that is predicted to be sufficient for a certain optimisation process in advance will both incur an inevitable monetary cost and cap the maximal possible number of islands. An alternative is to execute islands in so-called serverless manner, benefiting from the Function as a Service (FaaS) facilities available in major public clouds [35, 5].

FaaS allows a cloud to run code without prior VM provisioning or managing. The computational power scales automatically, is highly available and fault tolerant. The first request can initially see several seconds response time but is shorter than 1s for the subsequent requests[1]. Using such services is also

[1] `https://console.bluemix.net/docs/openwhisk/openwhisk\_compare.html`

economically beneficial as there is no charge for the time when code is not running. As FaaS automatically provides as much capacity as needed and a user is billed based on per-second capacity consumption and executions, it is suitable for performing distributed computation with several computing nodes.

It is then suitable for the Island Model of GA mentioned earlier.

Initially, FaaS was intended to execute short functions written in certain programming languages (for example JavaScript or Java). IBM was the first major public cloud vendor that permitted execution of Docker[2] containers as a function using IBM Cloud Function[3] based on Apache OpenWhisk[4]. However, by using open-source, Kubernetes[5]-native serverless framework named Fission[6], Docker containers can be executed as a function on any Kubernetes cluster, available in all major cloud facilities. These possiblities are discussed later in this paper.

The prospect of having a practically unbounded number of islands and the short provisioning time of a new island encourage to investigate the algorithms for dynamic managers governing the numbers of islands. In the manufacturing domain, these managers need to be effective for generating production plan and schedule for a wide range of smart factories. In particular, such managers should be appropriate to factories assembling discrete parts through mechanical processes (i.e. discrete manufacturing) and combining supplies or ingredients according to recipes (i.e. process manufacturing).

In this paper, three proposed managers are used for both the discrete and process manufacturing branches based on the real-world use cases provided by our business partners. As the multi-objective extensions of the managers known from the literature are shown to be successfully applicable only in certain situations (discussed in this paper), a new manager is proposed that combines the benefits of both the extensions and hence can be applied to assorted smart factories for determining the production plan and schedule. It is then suitable for the Island Model of GA mentioned earlier.

The main contribution of this paper can be summarised with the following points:

- proposing of multi-objective extensions of two managers determining dynamically the number of islands for single-objective GAs,
- critical analysis of the applicability of these two extensions,
- proposing of a new, more universal dynamic manager,
- describing a serverless cloud deployment of the Island Model of GAs,
- experimental evaluation of the proposed managers based on real-world discrete and process manufacturing scenarios.

[2]https://www.docker.com/
[3]https://console.bluemix.net/openwhisk/
[4]https://openwhisk.apache.org/
[5]https://kubernetes.io/
[6]https://fission.io/

A preliminary version of this work has been published in [38]. This paper extends that work by proposing the third, more universal manager than the two presented in [38] and providing an exhaustive critical analysis of the application of the three managers. The plan of the conducted experiments has been altered to evaluate all the three proposed approaches. The results have been evaluated using a larger set of quality indicators, reflecting both the solution set convergence and diversity.

The rest of this paper is organised as follows. After the brief survey of related works in Section 2, the general architecture of the developed system is presented in Section 3. The cloud deployment of the optimisation module is sketched in Section 4. The strategies for determining the number of islands are proposed in Section 5 and applied to real-world use cases in Section 6. Finally, the paper is concluded in Section 7.

## 2. Related Work

As this paper considers a dynamic number of GA islands in clouds, the prior research related to both the GA cloud execution and determining the number of islands is reviewed in this section.

One of the first research on executing GAs in clouds has been conducted by Di Martino et al. [4]. They proposed GA parallelisation at the fitness evaluation, population or individual levels applying the popular Map/Reduce programming model. The presented proof-of-concept implementation was performed using the Google App Engine web framework. It evaluated the fitness values of individuals in parallel and hence a significant overhead was reported. To decrease the communication overhead, the population-level parallelism (aka the Island Model) has been applied to the solution proposed in this paper.

Leclerc et al. in [18] proposed a framework for evaluating the fitness values in parallel across a heterogeneous pool of computing nodes. These nodes were both personal computers and VMs from a cloud vendor. Again, the master-slave architecture was applied. The nodes communicated with each other using JSON over HTTP. Due to the applied parallelism at the fitness evaluation level, the solution was reported to be beneficial only for expensive fitness evaluations. As in the proposed solution, the parallelisation is performed at the population level, it shall be beneficial for the less time-consuming fitness evaluations. Similarly, executing a number of GA iterations inside a slave node decreases the communication overhead caused by transferring the data using the rather verbose JSON data-interchange format.

The release of the Docker software in 2013 changed software deployment and orchestration in cloud systems. Since that time, the operating system virtualisation has gained popularity due to much higher performance and flexibility in comparison with a traditional, hypervisor-based virtualisation, offering sufficient isolation for numerous applications [3]. Initially, Docker containers were executed on a single machine, but soon a few of orchestration software managing a number of nodes in a cluster emerged, such as Docker Swarm or Google Kubernetes. For the first time, GA has been containerised using Docker in positional paper [30]. Its authors stressed that using containerisation facilities benefiting from one of the most attractive features of cloud computing which is the possibility of

on-demand resources allocation. This idea has been implemented in [29], where a conceptual workflow to support the development, deployment and execution of distributed GAs has been proposed. Following that workflow, a parallel GA has been employed where a master node executed the entire GA except for the fitness evaluation performed on slave nodes. In contrast to that approach, the proposed solution benefits from the population-level parallelism and uses the Kubernetes container-orchestration system for automating application deployment rather than CoreOS, used in that paper. In contrast to [29], the containers are executed in the serverless manner that enhances the scalability of the Dockerised GA optimisation process. In a serverless approach, no state needs to be maintained between invocations and, consequently, no VM is required to be instantiated in advance. The available computing resource is hence practically unbounded. The payment is made only for the real computation time of the computing resources.

Ma et al. employed the population-level parallelisation in [22]. Their solution followed the master-server architecture. The number of the slaves was decided statically. Each slave obtained a subpopulation of the size inversely proportional to its CPU utilisation. Then the corresponding fitness values were computed and returned to the master. In the proposed solution, the subpopulations have an equal size, but their number is changed dynamically during the optimisation process.

A simple proof-of-concept GA implementation in [8] applied the Island Model GA. The islands have been executed in the serverless manner which leverages the scaling capabilities of that solution. However, the number of islands was set statically and could not be altered during the optimisation process. No implementation details nor experimental results were provided to back the claims regarding the performance of that proposal.

Selecting an appropriate fixed number of islands is difficult and hence the methods that employ dynamic strategies for determining the number of islands have been proposed. During their run, the subpopulations are created and deleted, depending on the optimisation process state.

An idea to increase the number of subpopulations when the optimisation process is stuck and to decrease it when the globally best-found solution is frequently improved has been proposed for single objective optimization problems in [17]. In that paper, the number of islands has been doubled in each generation which has not improved the quality of the found solution. Otherwise, the number of islands has been either halved or decreased to one, depending on the applied algorithm. That technique has been shown, in a theoretical way, to reduce the expected computation time significantly compared to a panmictic (serial) GA. However, that approach is difficult to applied in practice as doubling the island number is not trivial in practice considering the limits of various cloud vendors (for example 20 EC2 instances in AWS EKS as in Spring 2019 and the initialisation time required by massive island creation can be significant, as a Linux instances were reported to be up and running as late as 90 seconds after launch [23]). That is the reason we followed strategies with a slower instance growth.

In [15], a Variable Island GA has been proposed where the initial number of islands has been set to one and each island has been populated by only two individuals. The decision of an island creation or removal has been taken based on

the status of convergence and the fitness of their two individuals. These resizing operations have been performed at each generation. The number of islands has been demonstrated to be rather stable and low during the entire minimisation process of benchmark multimodal continuous functions. That approach, however, would be difficult to be applied to multi-objective optimisation as covered in this paper since the total population size on all islands would be too low to offer required diversity of the solution space. Also, the overhead related to the frequent resizing operations would be rather significant in a cloud environment.

A Population-Merging Parallel Model for Evolutionary Algorithms has been proposed in [1]. In that model, a predefined number of islands is created at the beginning of the optimisation process. Then, the subpopulations evolve but no migration is performed. Periodically, a pair of islands whose entropy value maximises the diversity of the resulting island population is merged into one island and the best-fitted individuals survive on such merged island. The number of islands in that model is impossible to grow and the authors of that paper suggested splitting subpopulations with a high entropy value as future work.

The centralisation-clustering framework managing the number of islands has been proposed in [24]. With that framework, the optimisation process starts with a single island. After evolving a predefined number of populations, the individuals in that island are split into a set of new islands in a way that similar individuals are assigned to the same islands. Then, at another predefined generation, all the subpopulations in the islands are combined and split again into a certain number of new islands. That method has been demonstrated to maintain diverse subpopulations, yet the computational cost of similarity measurement or spectral clustering can be perceived as too high for the real-life industrial scenarios considered in this paper. In addition, the number of islands in the method is determined solely by the number of clusters and, hence, is unbounded. The subpopulations' sizes can be very different and hence, when executing in a cloud, the workload of particular instances would vary significantly.

The examples of strategies with bounded and rather slow islands' number growth are Classic [16] and Active [27] strategies, also for single objective problems. Both these strategies increase the number of subpopulations by one when all subpopulations are stuck. The difference between them is that the Classic strategy removes the subpopulation when there is another one investigating the same or a similar part of solution space size. The Active strategy removes all subpopulations except the one containing the global best solution when this solution is improved. In this paper, both these strategies are extended to multi-objective optimisation and used for real-world problems.

In [5], a GA has been parallelised at the population level and each subpopulation has been optimised by a Dockerised slave in accordance with the *serverless* computing paradigm. The optimisation process started with a predefined number of islands. In case an island had not improved the results for a certain number of GA iterations, it was removed from the pool. However, the main contribution of that paper was a new ending criterion based on the predicted profit rather than determining the number of islands in a dynamic manner. In this paper, a similar deployment scheme is applied, but the focus is on the determining of the number of islands.

## 3. System Architecture and Problem Description

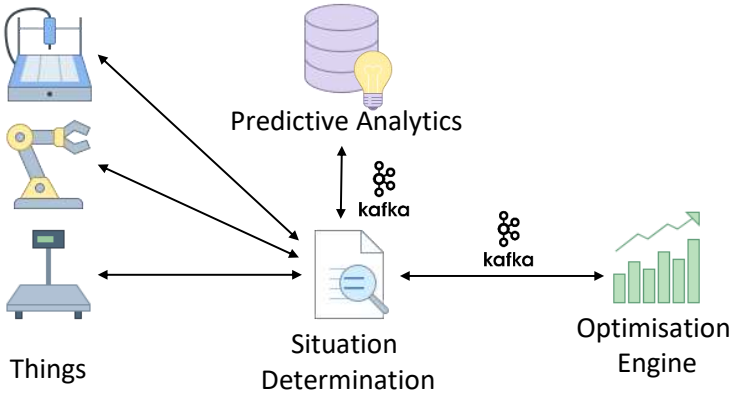


Figure 1: Overall architecture of the proposed system

The optimisation problems considered in this paper concern an integrated process planning and scheduling in smart factories, performed by the component named Optimisation Engine (OE). This component is a part of a larger project whose overall architecture is sketched in Figure 1. The scope of the entire project flow starts with data acquisition from various industrial devices (so-called things) using well-known protocols, according to the standards such as ISA95 or IEC 61449. This data is transferred to the Situation Determination (SD) module, which identifies the current state of the commodities, machines and/or processes. SD employs a specific use-case situation model based on an ontology similar to the one presented in [19]. The SD module monitors raw data provided by systems/sensors, which is also forwarded to the Predictive Analytics (PA) module. This module executes machine-learning-based methods to identify patterns in data to influence the decisions made by SD. When SD detects any relevant change of the factory state, including arrival of a new manufacturing order or a resource failure, the factory reconfiguration is performed by triggering the optimisation process performed by OE again. All the modules exchange information by employing Kafka[7], a high-throughput, low-latency platform for handling real-time data feeds. The messages sent between SD and OE follow a custom, well-defined textual protocol containing three types of numeric or enumerated values: (i) key objectives to be optimised, (ii) control metrics whose values can be altered to obtain various solutions and (iii) observable metrics, which contain other important information, for example unavailability of certain resources. Despite the appropriate functionality of both the SD and PA modules are critical for performing effective optimisation, the detailed descriptions of these modules are not in this paper scope.

The key capability of OE is the ability to respond to dynamic reconfiguration requests. Functionally, OE takes as input a configuration (an instantaneous

[7] https://kafka.apache.org/

description of the manufacturing process), issued by SD, and outputs a reconfiguration containing the proposed control values to be applied to the smart factory. The proposed reconfiguration is one that is of high quality, as determined by the objective function. This function is generated automatically based on a factory configuration specified by an XML-based format derived from the ontology used by SD.

## 4. Cloud deployment

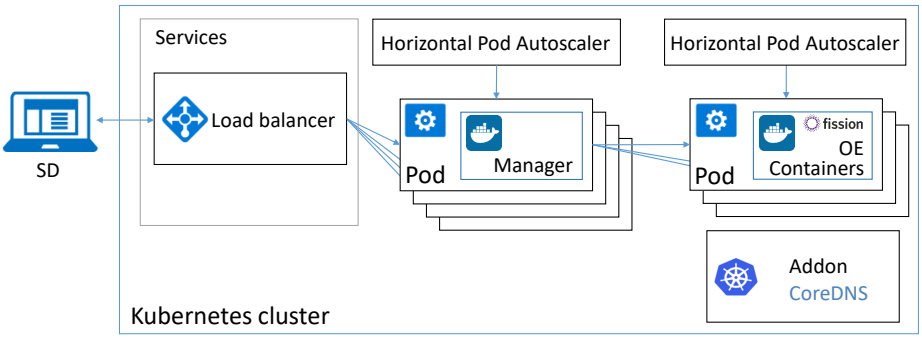


Figure 2: Kubernetes cluster for OE

OE is available as a Docker container, to be deployed in a cluster of the Kubernetes container-orchestration system. Kubernetes clusters are available in all major cloud facilities, including AWS Elastic Kubernetes Service (EKS), Azure, CloudStack, GCE, OpenStack, OVirt, Photon, IBM Cloud Kubernetes Service, as well as can be installed locally. Since OE is a stateless container whose optimisation function is executed by Kafka requests, it is suitable to use Fission, a framework for serverless functions on Kubernetes.

The services and pods (co-located containers) deployed in a Kubernetes cluster are presented in Figure 2. The number of OE containers is scaled automatically using the autoscaling facilities. This way a parallel optimisation process is performed by a distributed evolutionary algorithm. In this approach, each OE container executes one or more islands. Each island maintains its own subpopulation for GA-based search. The islands periodically exchange the best solutions found so far in a process called migration. The proposed algorithms determining the number of islands are presented in the following section. A new island is initiated by calling the main function of the OE container. Then the autoscaler provided by Fission decides whether the new island is to be maintained on the existing OE containers or a new container needs to be initiated. Not only the number of OE containers is steered automatically by the cloud services, but also the load balancing facility selects the least busy container to maintain a new island. In the current version of Fission, the autoscaler and load balancer are based solely on the CPU usage. In future, however, it is announced to include other metrics, such as based on memory footprint or network traffic.

The number of OE containers is dynamically changed to answer the reconfiguration process needs. In case of a higher demand caused by e.g. a need

for performing optimisation using a large number of islands, the number of containers is seamlessly scaled up to keep the appropriate computing power and similarly scaled down in case of lower demand. Such an approach not only guarantees the high system uptime, theoretically equal 100%, but also increases the responsiveness of OE and lowers the monetary cost in case of the deployment in a public cloud. The autoscaling feature is utterly transparent to the user. Even in an unlikely case of any container crash, the load balancer immediately switches the user to another container built from the same image in a way that no user data nor session details are lost. This feature is possible due to the fact that the OE containers are designed to be stateless, i.e., they do not store any session-related data that shall be persistent. Such a distributed, container-based architecture of the proposed solution is in line with the state-of-the-art software design and deployment, as mentioned in the Related Work section.

## 5. Dynamic determination of the number of islands

This section presents a cloud-based multi-objective optimisation using an Island Model of GA where the number of islands is determined dynamically by a master node named `manager`. Each island executes a single instance of OE for a fixed number of generations. In total, four versions of managers with various criteria for island creation and deletions have been developed: `ManagerStatic`, `ManagerClassic`, `ManagerActive` and `ManagerCalmActive`, where `ManagerStatic` provides the fundamental architecture of the proposed approach and is treated as the baseline approach. The remaining three algorithms introduce the dynamic islands management facility. The design rationale of `ManagerClassic` and `ManagerActive` is inspired by the single-objective managers presented in [27, 16].

### *5.1. `ManagerStatic` Strategy*

Algorithm 1 presents the pseudo-code for `ManagerStatic`. The algorithm starts by creating $N$ islands with OE which are executed using one or more OE containers, as decided by the Kubernetes cluster. Each island contains a typical GA optimiser with a randomly generated population of size $P$. The choice of the GA applied is user-defined, and using various GAs in different islands simultaneously is also possible. The manager maintains a global Pareto Front approximation ($PF$) that stores non-dominated solutions[8] reported by all the islands. The number of generations optimised during execution of each island is controlled by stage parameter $S$. In one stage, GA is executed for $I$ generations in each island. Then, the global $PF$ is updated with the results of all islands. Finally, the global $PF$ is returned as the output of the entire optimisation.

Instead of performing an isolated optimisation on each individual island during the entire optimisation process, a migration functionality is introduced to allow periodic information exchanges between all the islands at the end of each optimisation stage. After one stage execution of all islands, the manager iterates

[8]For a solution $s$ and a Pareto Front approximation $PF$, $s$ is said to be non-dominated if $s$ has a better value than any solutions in $PF$ for any objective.

**Algorithm 1:** Algorithm of `ManagerStatic`

**inputs** : $N$: initial number of islands;
$I$: number of iterations per stage;
$P$: number of individuals per island;
**outputs** : $PF$: a global Pareto Front approximation maintained by the manager;

1 $PF = \emptyset$, s = 0;
2 Create $N$ islands with $P$ randomly generated individuals;
3 **while** $\neg$ *stopCondition* **do**
4 | Execute all islands for $I$ iterations;
5 | Add non-dominated solutions returned from all islands into $PF$;
6 | Make migrations;
**end**
7 **return** $PF$;

through each island (starting from island 1) and randomly selects another island (e.g., island 3). Then, the best individual in island 1's population is selected and added to island 3's population by replacing the individual with the worst quality. Since any two islands can be selected as a source and destination of the migration process, the applied island topology is a fully connected graph. The approach for measuring the quality of individuals under multi-objectives can be various (e.g., weighted sum with normalisation) and is user-defined based on the specific optimisation problem. By default, we measure the general distance of individuals to select the individual that is the closest to the ideal point[9] for migrations. With the migrations applied, information from independent search processes conducted by various isolated islands can be shared and utilised by each individual optimisation process, which could improve the diversity of populations in all the islands, and hence, prevents a premature convergence.

### *5.2. `ManagerClassic` Strategy*

Based on the fundamental architecture of `ManagerStatic`, `ManagerClassic` introduces an island management mechanism so that the number of islands changes dynamically based on the temporal state of the optimisation process, as shown in Algorithm 2.

With `ManagerClassic`, the manager checks the quality of the global $PF$ at the end of each stage and compares it with that of the $PF$ in the previous stage (Line 7). Various comparator indicators (CI) are available for obtaining the quality for Pareto Front approximations and the choice is arbitrary. However, in this work, we assume a higher value returned by the applied CI represents higher quality. If the current global $PF$ has a higher CI value (i.e., $CI^{PF}$) than that of the previous one, it indicates that a better solution has been found in at least one island during the current stage and the Diversity Comparator Indicator (DCI) from [20] is used in proposed managers. Then the algorithm proceeds to the next stage. Otherwise (e.g., a local optimal point has been reached), the

[9]The ideal point is computed by optimizing each objective individually.

**Algorithm 2:** Pseudo-code of `ManagerClassic`

```
1  PF = ∅, s = 0;
2  Create N islands with P randomly generated individuals;
3  while ¬ stopCondition do
4  |   Execute all islands for I iterations;
5  |   Add non-dominated solutions returned from all islands into PF;
6  |   Make migrations;
7  |   if CI^PF is higher than in the previous iteration then
   |   |   continue;
   |   else
8  |   |   Delete islands that meet island deletion criteria;
9  |   |   Create one island with P randomly generated individuals;
   |   end
10 |   Return PF;
   end
```

manager removes an island if at least one from the criteria listed below is fulfilled (Line 8):

- a considered island has a population where all its individuals have the same genotype,
- there exists an identical population at another island,
- the population of a considered island is strictly dominated[10] by that of another island.

The above situations indicate that either the optimisation in an island is converged (i.e., the solutions found by each island are increasingly similar to each other) or the island is dominated by others and, hence, its population is not likely to improve the final result. As the island's population is unlikely to contribute to the finally reported optimum, it is dropped by the manager. Finally, the manager creates new islands to increase the total number of individuals in subpopulations and proceeds to the next optimisation stage[11]. Notice, in case of many-objective optimisation problems, most of the solutions may become non-dominated [31]. In such cases, a relaxed variant of Pareto dominance can be applied as an island removal criterion instead of the strict dominance, for example, partial Pareto dominance, where only a subset of objectives is considered [31]. Other relaxed variants of Pareto dominance can be also used, for examples $\alpha$-dominance, $\epsilon$-dominance or cone $\epsilon$-dominance, all compared in e.g. [2]. We plan to investigate the influence of various dominance relations on the optimisation process quality and cost in future.

With this algorithm, the quality of each island is monitored periodically and is handled dynamically based on the quality of its population. Intuitively, after

[10] For two Pareto Front approximations $PF_1$ and $PF_2$, $PF_1$ strictly dominates $PF_2$ if $PF_1$ contains at least one solution that has a better value than any solutions in $PF_2$ for all objectives.

[11] New islands are created with individuals generated randomly. This generation applies here and after for all island managers proposed in this paper.

applying this approach, islands are likely to have better quality than that of `ManagerStatic` and hence it could lead to a better optimisation performance. This claim is confirmed by the experiments shown later in this paper.

### *5.3. `ManagerActive` Strategy*

Similar to `ManagerClassic`, islands in `ManagerActive` are managed dynamically, but with different island deletion and creation conditions, as given in Algorithm 3.

**Algorithm 3:** Pseudo-code of `ManagerActive`

```
1  PF = ∅, s = 0;
2  Create N islands with P randomly generated individuals;
3  while ¬ stopCondition do
4  |   Execute all islands for I iterations;
5  |   Add non-dominated solutions returned from all islands to PF;
6  |   Make migrations;
7  |   if CI^PF is higher than in the previous iteration then
8  |   |   Delete all islands that do not provide new solutions to PF;
   |   else
9  |   |   Create one island with P randomly generated individuals;
   |   end
   end
10 return PF;
```

At the end of each stage, the manager compares the current $PF$ with the previous one via the assumed CI (Line 7). If the quality of the current $PF$ improves, it iterates through all the islands and deletes each island that does not provide new non-dominated solutions (Line 8). Otherwise (i.e., no islands can provide new solutions), it adds a new island to improve the exploration of the search space. Then the manager proceeds to the next optimisation stage.

Under `ManagerActive`, the islands are less stable than with the remaining algorithms as the manager either deletes a certain number of islands or adds a new one at each optimisation stage. The advantage of this approach is that once a local optimal point has been reached and the $PF$ quality stays the same for several stages, `ManagerActive` is able to maintain the highest number of individuals among the algorithms proposed earlier in this paper. Hence it is more likely to make further progress. However, in the case the global $PF$ is improved continuously, this approach deletes islands at each stage regardless of whether the island reached its local optimum or not. Consequently, the island is removed even if it could potentially improve the final solution at a later stage.

### *5.4. Properties of `ManagerClassic` and `ManagerActive`*

The effectiveness of a single-objective `ManagerClassic` has been discussed in [27]. From that discussion, it follows that `ManagerClassic` is less effective when a long period of a relatively easily achievable growth is observed after the method is stuck. In such the situation, `ManagerClassic` maintains too many subpopulations which hardly contribute to the final solution as the subpopulation with the easily achievable growth is likely to outperform the remaining subpopulations

in the following iterations. It is then reasonable to decrease the number of subpopulations to one and evolve only the most promising subpopulation. This tactic prevents from wasting computational power on evolving the remaining, less encouraging subpopulations. This situation is presented in Fig. 3. In that figure (and also in the remainder of this subsection), term *iteration* refers to a single manager iteration rather than a single iteration of GA. Thus, in each method iteration, subpopulations are processed during a user-defined number of GA generations.

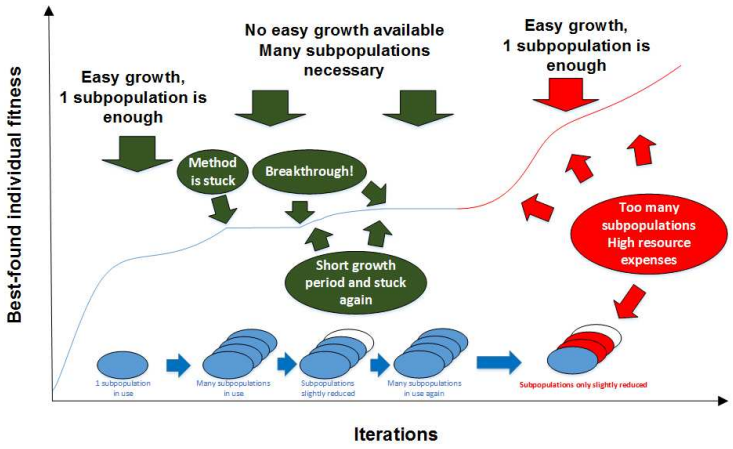


Figure 3: Example of a globally-best individual fitness over time in the Island Mode GA governed by `ManagerClassic`

Fig. 3 presents an example of a multi-population GA run where the number of islands is governed by `ManagerClassic`. In the beginning of the timeline, the globally best-found individual is improved at each method iteration. During that interval, a low number of subpopulations is sufficient to gain a continuous growth of the best-found individual. At a certain iteration, the method gets stuck. To escape from the local optima, the manager increases the number of subpopulations. This action increases the diversity of the individuals in these subpopulations and finally leads to a breakthrough. After the breakthrough, `ManagerClassic` may slightly reduce the subpopulation number. Then, after a short period of growth, the method is stuck again. However, since `ManagerClassic` has preserved most of the subpopulations, it is relatively easy to reach another breakthrough. After the second breakthrough, `ManagerClassic` preserves the majority of the subpopulations again. However, this time it is not beneficial as the method encounters a period of long and easy to find growth. At this point, the maintenance of many subpopulations wastes computational power.

The behavior of `ManagerActive` is similar to `ManagerClassic`. The only difference is removing all subpopulations except the one including the globally best individual after the fitness of that individual improves.

Fig. 4 presents an example of the globally-best individual fitness obtained during a multi-population GA-based optimisation process governed by `ManagerActive`. In the beginning of the timeline, the manager behaves similarly to

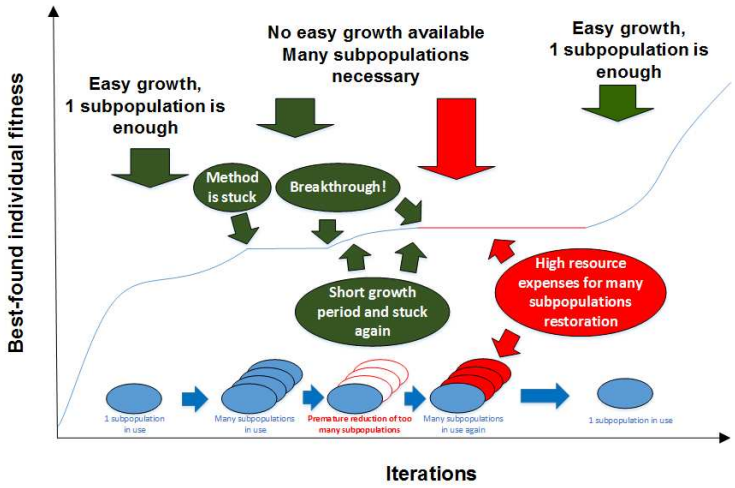


Figure 4: Example of the globally-best individual fitness over time in the Island Mode GA governed by `ManagerActive`

`ManagerClassic`. However, after reaching the first breakthrough, all subpopulations except the one containing the best individual are removed. Such a tactic appeared to be not beneficial due to the relatively short time of the following easy-improvement period. Therefore, when the method is stuck again, the subpopulation number is increased, which is rather costly. However, the same tactic is beneficial after the second breakthrough, as the improvement interval of the current globally-best individual is longer. Consequently, removing all subpopulations except the one containing the best-found individual is beneficial as it decreases the unnecessary computational cost of evolving the remaining islands.

### *5.5. `ManagerCalmActive` Strategy*

As discussed in the previous subsection, the situations in which one manager outperforms another are contradictory. Therefore, a new manager can be proposed that removes unnecessary subpopulations when they are less likely to influence the final solution (i.e., not after every single breakthrough). The manager shall behave appropriately in both the situations described above, namely: (i) it should avoid the premature deletions of the subpopulations without the globally-best individual found so far and (ii) it should not preserve too many unnecessary subpopulations. A manager with these properties is presented in Algorithm 4, and is later referred to as `ManagerCalmActive`.

As presented in Algorithm 4, the proposed `ManagerCalmActive`, behaves in the same way as `ManagerActive`, but it delays the deletion of subpopulations other than containing the best-found individual. The assumed delay interval is equal to the number of created subpopulations. For instance, if `ManagerCalmActive` had created eight subpopulations before a breakthrough was reached, all the subpopulations except the one containing the globally best individual are deleted if during subsequent eight iterations the fitness of the

**Algorithm 4:** Pseudo-code of `ManagerCalmActive`

```
1  PF = ∅, ImprInARowCounter = 0;
2  Create N islands with P randomly generated individuals;
3  while ¬ stopCondition do
4  |  Execute all islands for I iterations;
5  |  Add non-dominated solutions returned from all islands to PF;
6  |  Make migrations;
7  |  if CI^PF is higher than in the previous iteration then
8  |  |  ImprInARowCounter = ImprInARowCounter + 1;
9  |  |  if ImprInARowCounter is equal to the island number then
10 |  |  |  Delete all islands that do not provide new solutions to PF;
   |  |  else
11 |  |  |  Create one island with P randomly generated individuals;
12 |  |  |  ImprInARowCounter = 0;
   |  |  end
   |  end
13 return PF;
```

best-found individual has not been improved. This tactic scales automatically and does not suffer significant costs increase during long periods of easy-to-find growths. On the other hand, the manager does not prematurely delete the subpopulations when the best-found individual has not been improved recently. Despite its simplicity, the proposed manager leads to a significant results quality increase, as shown in Section 6.

## 6. Real-world Manufacturing Problems

Based on a number of problems originated in two real-world smart factory optimisation scenarios, this section presents experiments evaluating the proposed cloud-based GA optimisation approaches. To provide a fair comparison, a typical MOEA/D multi-objective genetic algorithm presented in [37] is applied to the OE, and the Diversity Comparator Indicator (DCI) from [20] is used where applicable to compare the quality of Pareto Front approximations for all the managers. The individual with the minimal makespan in a population will be selected for migration, which describes the manufacturing time of a commodity and is a crucial metric for both the studied real-world manufacturing use cases. For each MOEA/D solver, the size of its external population (which contains the resulting non-dominated solutions) is set to 100 and the size of neighbourhood is set to 30. The one-point crossover is applied to generate new individuals, with the mutation rate set to 0.1. Tournament selection is applied with a size of 3. The weights of the individuals are generated using the Hammersley's low-discrepancy sequence generator [10] and the fitness values are computed based on the Tchebycheff approach [25]. If not explicitly stated otherwise, parameter settings of $N = 5$, $N_{max} = 10$, $S = 40$, $G = 50$ and $I = 20$ are applied throughout this section. These values have been determined experimentally using a set of various optimisation problems [7, 38].

### *6.1. Graph-based comparison*

In this subsection, the quality of the results obtained with all the proposed managers is visualised in graphs. More detailed comparison, using various numeric indicators, are given in Subsection 6.2 by conducting these experiments under 40 test cases.

#### *6.1.1. Process Manufacturing Optimisation Problem*

The first real-world scenario is related to a manufacturing process for mixing/dispersion of powdery and liquid components, following a stored recipe. The main optimisation objective of this case study is to increase production line utilisation and, consequently, to decrease the makespan of batch production. The recipes can be executed on different compatible resources. Various recipes can be used to produce the same commodity. Consequently, the decision problem includes the selection of the multisubset (i.e. a combination with repetitions) of the recipes and their allocation to compatible resources, such that the appropriate amount of goods are produced with the minimal surplus in the shortest possible time.

The considered problem is characterised with multi-objective criteria, since not only the makespan needs to be minimised, but also the number of manufactured commodities should be as close to the ordered amounts as possible to minimise the storage costs.

The example factory consists of a set of mixers. There are five identical 5 tonne mixers, named Mixer 1-5 ($M_1$, $M_2$, $M_3$, $M_4$, $M_5$, respectively), and two identical 10 tonne mixers, named Mixer 6 ($M_6$) and Mixer 7 ($M_7$). There are two special 10 tonne mixers: Mixer 8 ($M_8$) and Mixer 9 ($M_9$). Four types of white paint can be produced in the factory and each mixer can be used to produce any commodity. However, the amount of paint produced during one manufacturing process and processing time vary depending on the mixer and paint types. For each combination of a mixer type and paint type, there is a unique recipe. The storage tanks, connected with the mixers via pipelines, limit the amount of the paints that can be produced as they have limited capacity. In case two recipes producing a different paint type are executed by the same mixer in sequence, a short sequence-dependent setup interval of the length provided by the business partner is enforced. A more detailed description of this scenario is presented in [7].

Table 1: Dynamic islands changes during one optimisation process execution process.

| Manager | Island Executions | Islands Created | Islands Deleted |
|---|---|---|---|
| `ManagerStatic` | 200 | 5 | 0 |
| `ManagerClassic` | 137 | 25 | 23 |
| `ManagerActive` | 192 | 29 | 19 |
| `ManagerCalmActive` | 339 | 11 | 1 |

Figure 5 shows the number of islands in subsequent stages during example execution of the optimisation process using the four managers described earlier. The number of islands creations, deletions and executions during these optimisation processes are given in Table 1.

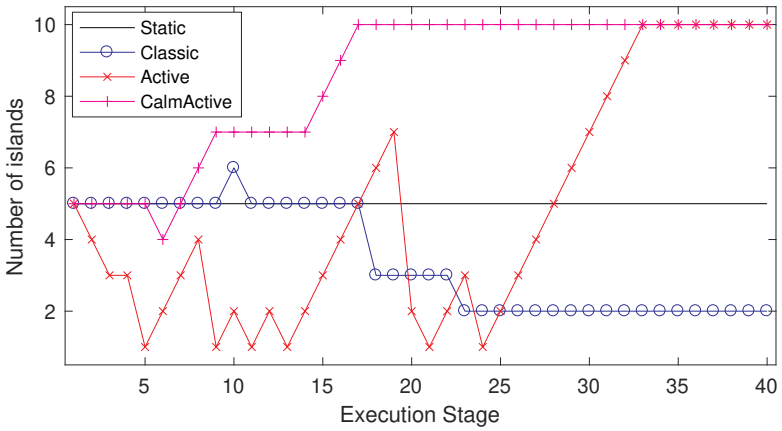


Figure 5: Number of dynamic islands during example execution of optimisation using different managers.

The `ManagerStatic` approach has the same number of islands during each optimisation stage, where the initial islands will be maintained and executed $S = 40$ times. For `ManagerActive`, this approach deletes islands that do not provide new globally non-dominated solutions (in case at least one island does so, see the 9th and the 20th stages) and keeps adding new islands as long as none can provide new non-dominated solutions at each stage (e.g., stages from the 13th to 19th). The figure demonstrates that the `ManagerActive` approach is converged at the 24th stage. After this stage, it keeps adding new islands and finally maintains 10 islands (i.e., the maximum allowed number of islands at one stage for all managers) at the 33rd stage and the later stages. Under `ManagerClassic`, the number of islands is decreasing in most stages. This approach firstly maintains the same number of islands in most stages before the 19th stage (where each island can provide new non-dominated solutions), but then the number of islands starts to decline. This indicates that a local optimal point has been reached and the algorithm is deleting islands that are less likely to contribute to the final solution while introducing new islands with random individuals. Finally, this approach is converged at the 23rd stage, where the algorithm deletes one existing island and then add a new one, as no island improves the quality of the global Pareto Front approximation. Notably, `ManagerClassic` performs the lowest number of island executions (137 in 40 stages) among all three algorithms but has the highest number of island deletions (i.e., the algorithm concludes that the optimisation has reached a local optimum). The number of islands maintained by `ManagerCalmActive` grows the fastest, but, as shown later in this paper, this manager is more efficient for a wider problem range than its competitors. Hence, an explicit trade-off between the optimisation quality and cost (but not time thanks to the distributed island execution in a cloud) can be observed.

The optimisation results for the considered process manufacturing optimisation problem is given in Figure 6, with a manufacturing order of 45, 40, 30,

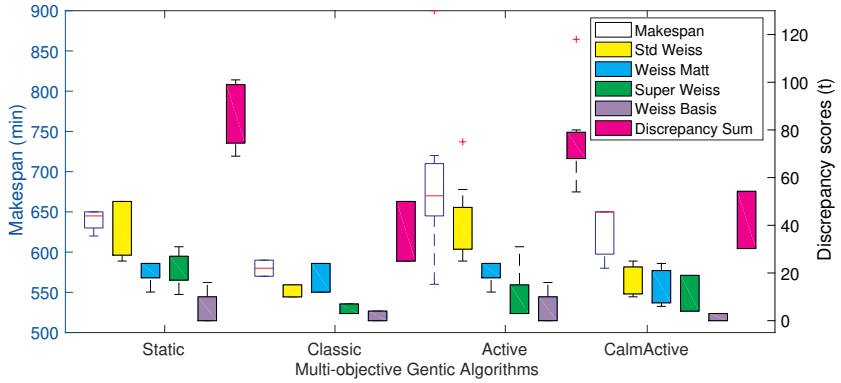


Figure 6: Process manufacturing optimisation results by all Managers

20 (in tonnes) of "Std Weiss", "Weiss Matt", "Super Weiss" and "Weiss Basis", respectively (the paint type names are in German). The makespan obtained using each manager is presented as a box with a blue frame and is associated with the blue Y-axis on the left-hand side of the figure. Commodity surpluses are denoted with the discrepancy scores (boxes with black frames), which are associated with the Y-axis on the right-hand side of the figure. Finally, the magenta box gives the sum of discrepancy scores of all commodities for each solution. As given in the figure, the `ManagerStatic` approach is outperformed by all three dynamic managers which provide a better minimisation for all the objectives. This is expected given the fact that this approach has a larger number of individuals (5 islands with 50 individuals, 250 individuals in total). However, although `ManagerActive` has the largest number of individuals ($25 \times 50$), its optimisation results are outperformed by `ManagerClassic`, which has a similar number of individuals but requires significantly fewer island executions.

The experiment has been conducted using Amazon EKS service in AWS London zone. The EKS cluster has been composed of m5.large EC2 instances, each including 8GiB RAM and 2 virtual CPU cores equivalent to 3.1 GHz Intel Xenon Platinum series cores. In case of `ManagerClassic`, maximum 4 EC2 instances have been used during the optimisation process, whereas `ManagerActive` and `ManagerCalmActive` used as many as 6 EC2 instances during 8 and 24 (out of 40) execution stages, respectively. On average, the number of m5.large EC2 instances has been equal to 2.5, 3.2 and 5.1 for `ManagerClassic`, `ManagerActive` and `ManagerCalmActive`, respectively. During the entire process, `ManagerStatic` has used 3 EC2 instances. In general, the prices paid for performing each of the above optimisation processes are less than 30$, where `ManagerClassic` incurs the least cost due to its well-controlled number of islands (i.e., 11$ against 15$ for `ManagerStatic`, 12.1$ for `ManagerActive` and 25.4$ for `ManagerCalmActive`).

### *6.1.2. Discrete Manufacturing Optimisation Problem*

The second considered real-world use case is based on the discrete manufacturing process of Wire-cut Electrical Discharge Machining (WEDM), in which the desired shape of a metal part is obtained by removing unnecessary material

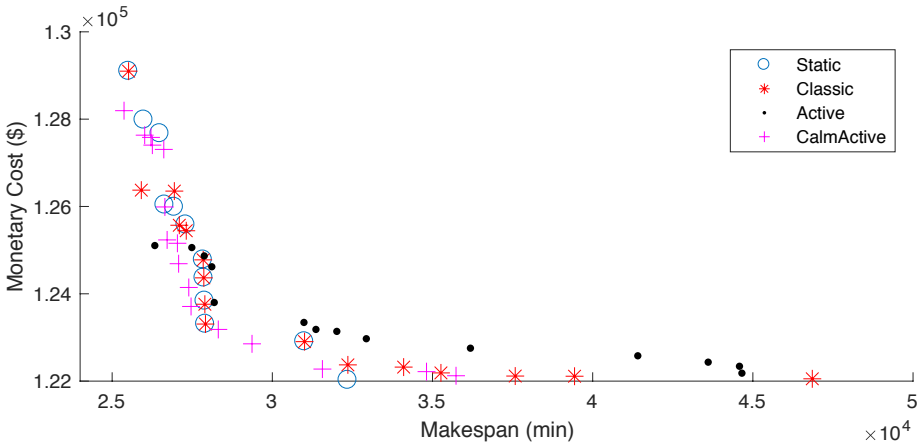


Figure 7: Pareto front approximations for the considered discrete manufacturing scenario

via a series of current discharges between two electrodes. One of these electrodes is a wire that is wound between two spools to avoid its erosion. The wire is the most expensive consumable in the process. Selecting a machine whose work cost is lower and, if possible, applying an "eco mode", can decrease the cost per part significantly. More details about the considered scenario have been presented in [6].

In the considered factory, a plant includes three WEDM machines: "small", "medium" and "large". The cost of machine usage per hour grows with the machine size. These machines are associated with various sizes of parts. Small parts can be manufactured on any machine, medium parts require a medium or large machine, and large parts are manufacturable only on large machines. All parts can be manufactured in one from four Manufacturing Ways (MW) that differ in wire types and machine modes.

The goal of this problem is to assign the part to machines, select its MW and schedule the production to decrease both the total cost and the overall makespan. Unlike the previous case, whose optimisation objectives are not necessarily conflicting with each other, objectives in the considered discrete manufacturing case (i.e., makespan and monetary cost) are difficult to be minimised at the same time, as a short makespan usually indicates using a larger machine and hence a higher monetary cost. Table 2 gives an example of the configuration for one manufacturing part $P1$, which can be produced via 12 different approaches (i.e., machines with MWs) with various cutting time and monetary costs. In the considered scenario, 16 various parts are ordered for production and each part will be produced once.

Figure 7 presents the optimisation results using the proposed algorithms with the following parameters: $N = 5$, $N_{max} = 10$, $S = 40$, $G = 50$ and $I = 20$. As shown in this figure, each proposed algorithm obtains a set of solutions evenly distributed on the Pareto Front approximation. The approximation returned by `ManagerStatic` has lower ranges with respect to both the objectives than the results of both `ManagerActive` and `ManagerClassic`. In addition, we ob-

Table 2: Example configuration of a manufacturing part for Discrete Manufacturing Optimisation Problem

| Part | Machine size | Manufacturing Way | Cutting time (min) | Wire cost per part ($) | Machine cost per part ($) | Total cost per part ($) |
|---|---|---|---|---|---|---|
| $P1$ | Small | 1 | 2833.5 | 28.1 | 164.0 | 192.1 |
| | Small | 2 | 2956.2 | 28.1 | 140.3 | 168.4 |
| | Small | 3 | 3042.1 | 28.1 | 147.8 | 175.9 |
| | Small | 4 | 3174.1 | 30.2 | 136.8 | 167.0 |
| | Medium | 1 | 2033.5 | 30.2 | 242.9 | 273.1 |
| | Medium | 2 | 2156.2 | 30.2 | 208.4 | 238.6 |
| | Medium | 3 | 2242.1 | 41.0 | 196.1 | 237.1 |
| | Medium | 4 | 2674.1 | 41.0 | 189.0 | 230.0 |
| | Large | 1 | 1256.2 | 41.0 | 555.9 | 596.9 |
| | Large | 2 | 1633.5 | 53.7 | 465.6 | 519.3 |
| | Large | 3 | 1842.1 | 53.7 | 427.9 | 481.6 |
| | Large | 4 | 1974.1 | 53.7 | 408.4 | 462.1 |

served that the `ManagerActive` approach obtains a Pareto Front approximation with the widest range in both the objectives. Unlike the previous case, where all objectives could be minimised at the same time and `ManagerClassic` outperformed `ManagerActive`, in the considered case algorithm `ManagerActive` delivers better results. Such a phenomenon is due to the nature of the considered optimisation problem, where the objectives contradict each other so that having a larger number of individuals in the entire population is more beneficial for the optimisation process.

During this optimisation, the islands of each manager follow the trend shown in Figure 5 in general, where islands with `ManagerActive` are created (or removed) rapidly and then increased up to 10 when finished. The number of islands in `ManagerClassic` remains stable and then decreases gradually in later stages. In addition, due to the relative simplicity of the objective function of the considered problem, the computation cost for performing this optimisation is much lower than that of the considered process manufacturing optimisation problem and does not exceed 0.5$ in general for all managers using the same cloud instances as earlier.

In the next experiment, the scalability of the proposed cloud-based optimisation algorithms is investigated. Scale factor $i = 1, \ldots, 10$ is introduced to control the size of the optimisation problem in a way that the number of ordered parts, available machines and manufacturing ways are multiplied $i$ times. For instance, with $i = 3$, part $P1$ will be produced 3 times with 36 possible selections of machines and manufacturing ways. The cutting time and costs are consistent with the values given in [6]. Using the scale factor, thirty test cases have been generated. The use of `ManagerActive` has led to higher quality results than `ManagerClassic` in 28 cases. Similarly, the GA employing `ManagerClassic` was better than the one using `ManagerStatic` in 27 runs. This leads to the following conclusions. For the considered problem, GA employing `ManagerActive` is more effective than with `ManagerStatic` or `ManagerClassic`. All GAs using the dynamic subpopulation number managements significantly outperform the one

with the static number of islands. These conclusions were confirmed by Sign Test with the probability exceeding 99%.

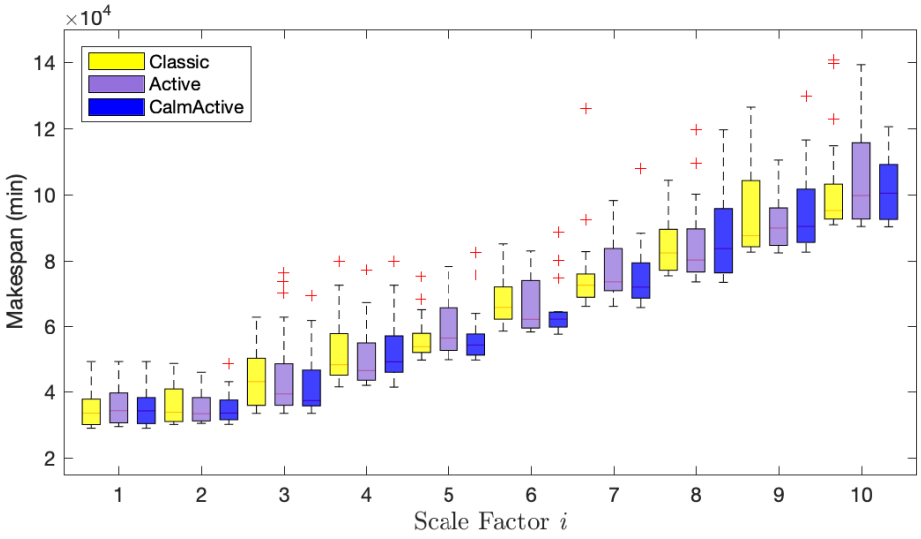


Figure 8: Makespan optimisation results of the proposed cloud-based approaches by scaling the problem size

Figures 8 and 9 present the optimisation results of makespan and total monetary cost, respectively, for the considered problem sizes. As given in Figure 8, optimisation results of makespan of all the proposed algorithms demonstrate a slowly increasing trend, which implies the majority of the parts are produced in parallel. Among the evaluated algorithms, `ManagerStatic` is outperformed by others in all cases, where values of makespan obtained by `ManagerStatic` have a lower range than those of `ManagerActive`, `ManagerClassic` and `ManagerCalmActive`. In addition, the `ManagerActive` approach delivers Pareto Front approximations with a higher diversity (for objective "makespan") than that of `ManagerClassic` in general, and this phenomenon is more visible for scaling factor $i \geq 7$. This observation again confirms the claim that algorithms with a larger population would benefit in solving such problems. In contrast, objective "monetary cost" in Figure 9 presents a linearly increasing trend. This is unavoidable as manufacturing each part is incurred with a certain cost. Such observation again confirms that optimisation results remain predictable while scaling the factory size.

Finally, due to the relatively low computationally demanding objective function, performing optimisation of a scaled factory size does not incur a huge cost for using cloud computing service. With $i = 10$, it takes 907s in average to execute a single stage, and the entire optimisation process costs below 10$ on all analysed AWS instance types of ECU[12] between 13 to 68.

[12] 1 ECU is defined by Amazon as the compute power of a 1.0-1.2GHz of server CPU from 2007.

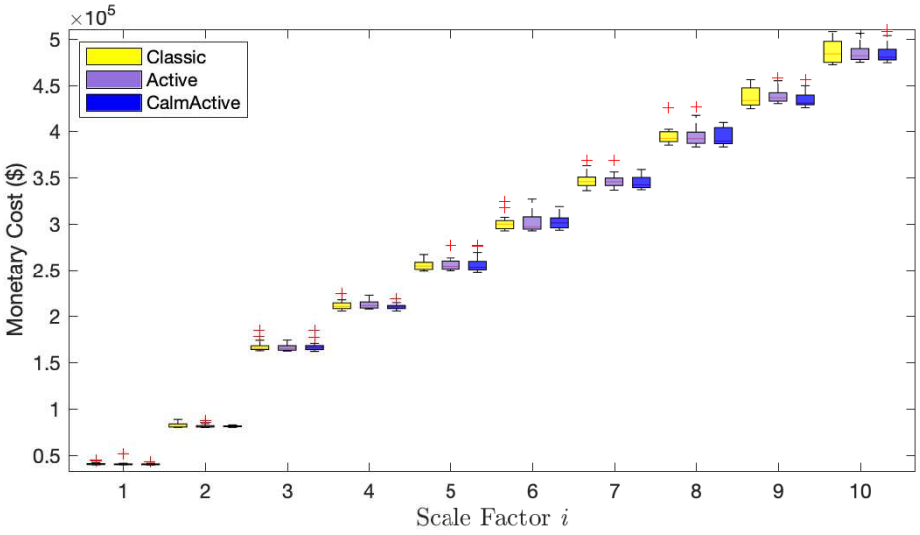


Figure 9: Monetary cost optimisation results of the proposed cloud-based approaches by scaling the problem size

### *6.2. Numeric comparisons*

In order to compare the managers more accurratelly, an additional experiment has been conducted. In this experiment, the testing cases have been split into two separate groups, related to both the process and discrete manufacturing optimisation problems described earlier. Each group consisted of 40 different test cases. The limit of 40 stages was used as a stop condition. To assure the fairness of the comparison, all methods shared all the possible parts of source codes. Thus, the source codes quality should not influence the reliability of the comparison.

To compare the results quality of the competing methods we use three different numerous quality indicators which are popular in the multi-objective optimization:

- Generational Distance (GD) [13],
- $D1_R$ [11],
- Diversity Comparator Indicator (DCI) [20].

The first indicator quantifies the convergence of the solution set, whereas the remaining two indicators quantify the diversity of the solution set. The simultaneous usage of the GD and $D1_R$ indicators to asses a front quality has been proposed in [14]. Two of these indicators (GD and $D1_R$) have been listed in the top 10 list of the most used metrics in evolutionary multi-criterion optimization according to [28], whereas DCI is more appropriate for many-criterion optimisation [20].

For each indicator, the ranking has been constructed in the following manner. The best method for a particular experiment receives the number of points equal

to the number of competing methods. The second best method receives one point lower, etc. If more than one method assumes the same score, they both receive the same number of points. For example, let us consider a case of four competing methods: A, B, C and D. Assuming that method A takes the first place, it receives 4 points. If methods B and D both take the second place, they both receive 3 points. If method C takes the last place, it receives 1 point. For all the competing managers, the average and median ranking value depending on experiment group are presented in Table 3.

Table 3: Average and median ranking for the considered Pareto front quality indicators

| | | **Process Manufacturing Optimisation Problem** | | | |
|---|---|---|---|---|---|
| | | **Static** | **Classic** | **Active** | **CalmActive** |
| **DCI** | **Average** | 2.45 | **3.13** | 1.85 | **2.88** |
| | **Median** | 2 | 3 | 1 | 3 |
| **GD** | **Average** | 2.45 | **2.78** | 1.85 | **2.73** |
| | **Median** | 3 | 3 | 1 | 3 |
| **D1**$_R$ | **Average** | 2.15 | **2.58** | 1.78 | **2.33** |
| | **Median** | 2 | 3 | 1 | 2 |
| | | **Discrete Manufacturing Optimisation Problem** | | | |
| | | **Static** | **Classic** | **Active** | **CalmActive** |
| **DCI** | **Average** | 2.23 | 2.48 | 2.60 | **3.23** |
| | **Median** | 2 | 2 | 3 | 4 |
| **GD** | **Average** | 2.58 | 2.20 | 2.35 | **2.80** |
| | **Median** | 3 | 2 | 2 | 3 |
| **D1**$_R$ | **Average** | 2.10 | 2.20 | 2.25 | **2.90** |
| | **Median** | 2 | 2 | 2 | 3 |

The motivation for `ManagerCalmActive` development was to propose a manager that would report the results of quality that is close to `ManagerClassic` for the experiments similar to the presented process manufacturing problem, and close to `ManagerActive` for the experiments related to the presented discrete manufacturing problem. The results reported in Table 3 show that `ManagerCalmActive` is the best for all indicators when the discrete manufacturing experiments are considered, but seems to be the second best for the process manufacturing experiments. However, the results reported by Sign-test show that there is no statistically significant difference in results quality between `ManagerClassic` and `ManagerCalmActive` for the process manufacturing experiments for experiment quality indicators. On the other hand, the same statistical test shows that for the discrete manufacturing experiment group for two statistical indicators `ManagerCalmActive` reports significantly better results than `ManagerActive`. The detailed $p$-values reported by Sign Test are reported in Table 4. Note that for the standard 5% significance level, only tests for DCI and $D1_R$ in the discrete manufacturing group are decisive. For the process manufacturing group, and the comparison of `ManagerCalmActive` and `ManagerClassic`, the $p$-values related to hypothesis 'the median result quality is equal' are higher. Thus, we can state that proposed `ManagerCalmActive` has fulfilled its objectives - it reports the results of quality that is at least equal to the results quality of

`ManagerClassic` for the process manufacturing group and at least equal to the results quality of `ManagerActive` for the discrete manufacturing group. Note that proposed `ManagerCalmActive` has actually exceeded these expectations and it reports results that are significantly better than the results of `ManagerActive` for two indicators in the discrete manufacturing group.

Finally, a traditional (panmictic) MOEA/D GA with no islands has been implemented, deployed to a cloud and tested against the proposed island-based optimisation methods. This method has a population size of 250 and 800 iterations (identical to 200 island executions in total). In general, the panmictic version is outperformed by the proposed methods under each applied quality indicator for both manufacturing scenarios in 40 different test cases with confidence level exceeding 99%. For both manufacturing scenarios, $[1.0, 0]$ is obtained by comparing each dynamic island-based manager and the panmictic version under DCI in 40 and 39 test casesl, respectively. In addition, the panmictic version consumes about 5 times more computation time (approximately 6.2 hours for process manufacturing and 24.2 minutes for discrete manufacturing) than that of the proposed methods.

Table 4: The $p$-values reported by Sign Test for results quality comparisons between `ManagerCalmActive`, `ManagerClassic` and `ManagerActive`

| | | `ManagerCalmActive` vs `ManagerClassic` | | |
|---|---|---|---|---|
| | | **better or equal** | **equal** | **worse or equal** |
| Process manufacturing | **DCI** | 0.31 | 0.62 | 0.80 |
| | **GD** | 0.57 | 1.00 | 0.57 |
| | **D1**$_R$ | 0.18 | 0.36 | 0.90 |
| | | `ManagerCalmActive` vs `ManagerActive` | | |
| | | **better or equal** | **equal** | **worse or equal** |
| Discrete manufacturing | **DCI** | 1.00 | 0.00 | 0.00 |
| | **GD** | 0.96 | 0.15 | 0.08 |
| | **D1**$_R$ | 1.00 | 0.01 | 0.00 |

### *6.3. Scalability and Run-time Cost*

The above experiments have been preformed with a fixed setting of $N = 5$ and $N_{max} = 10$. In this subsection, these parameters are increased to evaluate the run-time costs of the proposed optimisation methods with a larger number of islands. Table 5 presents the average run-time cost of various operations in the proposed island-based optimisation methods for both the considered manufacturing scenarios for scaled values of $N_{max}$ (and $N = \frac{N_{max}}{2}$). Each value in this table is an average from 50 executions of `ManagerActive` (i.e., the manager that experiences the most frequent island creation and removal operations). As shown in the table, the island execution time of each stage is not affected by the growing number of islands. It takes about 104 s to execute one stage for the process manufacturing and 9 s for the discrete manufacturing optimisation. Besides, maximum one island can be created in each stage and hence its cost is stable while increasing $N_{max}$. However, the cost of island removal and gene migrations demonstrates an increasing trend with the growth

Table 5: Computation cost of operations in each execution stage in `ManagerActive`

| $N_{max}$ | 10 | 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|---|---|
| GA execution (process) | 103.79 s | 102.86 s | 103.19 s | 102.85 s | 103.87 s | 105.23 s |
| GA execution (discrete) | 8.76 s | 9.07 s | 8.72 s | 8.61 s | 9.64 s | 8.64 s |
| island creation | 0.25 s | 0.18 s | 0.18 s | 0.15 s | 0.13 s | 0.18 s |
| island removal | 0.16 s | 0.39 s | 0.68 s | 0.96 s | 1.25 s | 1.60 s |
| one migration | 0.03 s | 0.05 s | 0.07 s | 0.10 s | 0.12 s | 0.15 s |

of the number of islands and it takes up to 1.6 s, and 0.15 s, respectively, for $N_{max} = 60$. This growth is caused by the manager needs to iterate through all the islands to determine the ones to be removed and to perform gene migrations.

For the considered values of $N_{max}$ and $N$, the cost for synchronising islands and migrations is relatively low compared to the cost for one optimisation stage. Thus, the total optimisation times of all the managers are dominated by the computation time of the GA iterations performed in the execution stages, especially in the process manufacturing case.

Table 6: DCI and GD measure of the proposed managers with varied $N_{max}$ under the discrete manufacturing optimisation problem

| $N_{max}$ | | 10 | 20 | 30 | 40 | 50 |
|---|---|---|---|---|---|---|
| Classic | **DCI** | 0.143 | 0.143 | 0.238 | 0.25 | 0.286 |
| | **GD** | 124157.8 | 124019.8 | 124114.2 | 124122.7 | 123997.8 |
| Active | **DCI** | 0.04 | 0.16 | 0.28 | 0.2 | 0.4 |
| | **GD** | 124106.7 | 124884.6 | 125928.9 | 124371.3 | 124566.9 |
| CalmActive | **DCI** | 0 | 0.138 | 0.276 | 0.276 | 0.31 |
| | **GD** | 124456.3 | 123931.0 | 124288.3 | 124102.9 | 123997.8 |

At last, Table 6 presents the quality measure (by DCI and GD) for comparing the optimisation results of each proposed manager via scaling $N_{max}$, based on the discrete manufacturing problem with a problem size $i = 3$. As shown in the table, increasing $N_{max}$ improves the diversity (i.e., the DCI measure) of the Pareto Front approximation obtained by each manager, where more evenly distributed solutions can be found with a larger $N_{max}$. As for the distance to the ideal point, under the giving optimisation problem, the resulting GD measures are similar for each manager under all $N_{max}$ settings. This observation demonstrates the optimisation can be effectively converged (i.e., reaching a local optimum for each objective) with the given configuration in the above experiments (i.e., with $N = 5$ and $N_{max} = 0$), and so for $N_{max}$ with larger values.

*6.4. Comparison with experts*

The business partner defining the process manufacturing optimisation problem has been gathering real production data for over a decade. It is then possible to apply the proposed approach to real historic manufacturing orders and compare the outcome with the decisions of a human expert operator. The related data has been extracted by the business partner for a randomly selected day from the considered factory past. Then, the OE configured in the way described earlier has been executed. As a result, the usage time of the mixers needed to produce the ordered amount of commodities has been decreased by about 13% and the production line usage has been better balanced. The latter observation can be backed with the comparison of standard deviation for the historical and optimised schedule which decreased by above 61% (i.e., about 4 hours).

The business partner defining the discrete manufacturing problem has not provided us with historic data but defined four simple heuristics that are typically followed by human operators instead. These heuristics have been implemented and then compared with OE for a randomly chosen historic part order. As the heuristics lead to a single solution rather than a Pareto set as in OE, the multi-objective problem has been transformed into a single-objective one by a weighted average of both objectives. Depending on the heuristics, OE returned a better solution from 10% to 18% in terms of the makespan, whereas the total cost was lower from 4% to 14%.

In conclusion, the proposed system has performed better than human experts in the analysed samples. Unfortunately, performing a comparison using a larger set of historical data is difficult due to the data confidentiality.

## 7. Conclusion and future work

This paper describes three cloud-based optimisation algorithms using dynamic manager-islands architecture. The proposed algorithms are then applied to real-world process planning and scheduling problems for two manufacturing plants representing different manufacturing branches. The optimisation engine for such problems and cloud deployment methods for the proposed optimisation approach are explained. The obtained experimental results have demonstrated the applicability and efficiency of the dynamic reconfiguration of resource allocation and scheduling for smart factories as well as their superiority over the traditional approach with a fixed number of islands. From the results, `ManagerClassic` seems to be more favourable where objectives do not conflict with each other (i.e., can be minimised at the same time) while `ManagerActive` is beneficial for the optimisation problems that require more individuals during each stage to list a larger number of possible solutions for conflicting objectives. `ManagerCalmActive` is performing relatively well in both these situation and hence can be preferable when the characteristics of the underlying optimisation problem are not well explored. As for `ManagerStatic`, this algorithm is suitable for cloud suppliers where creating (or removing) a distributed note is costly (in terms of time or money).

Besides the manager-islands architecture adopted in this paper, other island topologies can be used instead of the fully connected graph, e.g. a ring. This would require executing an algorithm that forms such topologies out of the

OEs and keeping the adjacency list in a database. In addition, some well-known distributed system algorithms to guarantee consistency and robustness (e.g. failure detectors) of the proposed system would be desirable for cloud-based computing. Further, for the recipe scheduling problems described in this paper, certain problem-specific islands management model and GAs could further improve the optimisation results. Applying a relaxed version of Pareto dominance as a criterion for island removal is likely to decrease the number of islands and, consequently, lower the optimisation cost. Finally, the impact of various quality indicators towards the optimisation results of the proposed algorithms will be investigated in future.

**Acknowledgment**

The authors acknowledge the support of the EU H2020 SAFIRE project (Ref. 723634).

**References**

[1] Arellano-Verdejo, J., Godoy-Calderon, S., Alonso-Pecina, F., Arenas, A. G., Cruz-Chavez, M. A., 2017. A new efficient entropy population-merging parallel model for evolutionary algorithms. International Journal of Computational Intelligence Systems 10, 1186–1197.
URL `https://doi.org/10.2991/ijcis.10.1.78`

[2] Batista, L. S., Campelo, F., Guimaraes, F. G., Ramirez, J. A., June 2011. A comparison of dominance criteria in many-objective optimization problems. In: 2011 IEEE Congress of Evolutionary Computation (CEC). pp. 2359–2366.

[3] Chung et al., M., July 2016. Using Docker in high performance computing applications. In: 2016 IEEE Sixth International Conference on Communications and Electronics (ICCE). pp. 52–57.

[4] Di Martino, S., Ferrucci, F., Maggio, V., Sarro, F., 2012. Towards migrating genetic algorithms for test data generation to the cloud. Software Testing in the Cloud: Perspectives on an Emerging Discipline, 113–135.

[5] Dziurzanski, P., Swan, J., Indrusiak, L. S., 2018. Value-based manufacturing optimisation in serverless clouds for industry 4.0. In: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '18. ACM, New York, NY, USA, pp. 1222–1229.

[6] Dziurzanski, P., Swan, J., Indrusiak, L. S., Ramos, J., 2019. Implementing digital twins of smart factories with interval algebra. In: IEEE International Conference on Industrial Technology. ICIT 2019. pp. 941–948.

[7] Dziurzanski, P., Zhao, S., Swan, J., Indrusiak, L., Scholze, S., Krone, K., 2019. Solving the multi-objective flexible job-shop scheduling problem with alternative recipes for a chemical production process. In: Applications of Evolutionary Computation - 22nd International Conference. EvoApplications 2019. pp. 33–48.

[8] García-Valdez, J.-M., Merelo-Guervós, J.-J., 2018. A modern, event-based architecture for distributed evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. GECCO '18. ACM, New York, NY, USA, pp. 233–234.

[9] Geyer, R., 2012. Identifying workloads for the cloud.
URL https://www.rightscale.com/blog/enterprise-cloud-strategies/identifying-workloads-cloud

[10] Hammersley, J., 2013. Monte carlo methods. Springer Science & Business Media.

[11] Hapke, M., Jaszkiewicz, A., Słowiński, R., Aug 2000. Pareto simulated annealing for fuzzy multi-objective combinatorial optimization. Journal of Heuristics 6 (3), 329–345.
URL https://doi.org/10.1023/A:1009678314795

[12] He, W., Xu, L., 2015. A state-of-the-art survey of cloud manufacturing. International Journal of Computer Integrated Manufacturing 28 (3), 239–250.
URL https://doi.org/10.1080/0951192X.2013.874595

[13] Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y., 2015. Modified distance calculation in generational distance and inverted generational distance. In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C. C. (Eds.), Evolutionary Multi-Criterion Optimization. Springer International Publishing, Cham, pp. 110–125.

[14] Ishibuchi, H., Tsukamoto, N., Nojima, Y., Sep. 2007. Iterative approach to indicator-based multiobjective optimization. In: 2007 IEEE Congress on Evolutionary Computation. pp. 3967–3974.

[15] Jumonji, T., Chakraborty, G., Mabuchi, H., Matsuhara, M., Sep. 2007. A novel distributed genetic algorithm implementation with variable number of islands. In: 2007 IEEE Congress on Evolutionary Computation. pp. 4698–4705.

[16] Kwasnicka, H., Przewozniczek, M., 2011. Multi population pattern searching algorithm: A new evolutionary method based on the idea of messy genetic algorithm. IEEE Trans. Evolutionary Computation 15, 715–734.

[17] Lässig, J., Sudholt, D., 2011. Adaptive population models for offspring populations and parallel evolutionary algorithms. In: Proceedings of the 11th Workshop Proceedings on Foundations of Genetic Algorithms. FOGA '11. ACM, New York, NY, USA, pp. 181–192.

[18] Leclerc, G., Auerbach, J. E., Iacca, G., Floreano, D., 2016. The seamless peer and cloud evolution framework. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference. ACM, pp. 821–828.

[19] Lemaignan, S., Siadat, A., Dantan, J. ., Semenenko, A., June 2006. Mason: A proposal for an ontology of manufacturing domain. In: IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06). pp. 195–200.

[20] Li, M., Yang, S., Liu, X., Dec 2014. Diversity comparison of pareto front approximations in many-objective optimization. IEEE Trans. on Cybernetics 44 (12), 2568–2584.

[21] Lissovoi, A., Witt, C., 2015. On the utility of island models in dynamic optimization. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. GECCO '15. ACM, New York, NY, USA, pp. 1447–1454.
URL http://doi.acm.org/10.1145/2739480.2754734

[22] Ma, N., Liu, X.-F., Zhan, Z.-H., Zhong, J.-H., Zhang, J., 2017. Load balance aware distributed differential evolution for computationally expensive optimization problems. In: GECCO Proceedings Companion, 2017. ACM, pp. 209–210.

[23] Mao, M., Humphrey, M., June 2012. A performance study on the vm startup time in the cloud. In: 2012 IEEE Fifth International Conference on Cloud Computing. pp. 423–430.

[24] Meng, Q., Wu, J., Ellis, J., Kennedy, P. J., May 2017. Dynamic island model based on spectral clustering in genetic algorithm. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 1724–1731.

[25] Miettinen, K., 2012. Nonlinear multiobjective optimization. Vol. 12. Springer Science & Business Media.

[26] Phanden, R. K., Jain, A., Verma, R., 2011. Integration of process planning and scheduling: a state-of-the-art review. International Journal of Computer Integrated Manufacturing 24 (6), 517–534.
URL https://doi.org/10.1080/0951192X.2011.562543

[27] Przewozniczek, M., Aug 2016. Active multi-population pattern searching algorithm for flow optimization in computer networks - the novel coevolution schema combined with linkage learning. Inf. Sci. 355 (C), 15–36.

[28] Riquelme, N., Von Lücken, C., Baran, B., 2015. Performance metrics in multi-objective optimization. In: 2015 Latin American Computing Conference (CLEI). IEEE, pp. 1–11.

[29] Salza, P., Ferrucci, F., 2019. Speed up genetic algorithms in the cloud using software containers. Future Generation Computer Systems 92, 276 – 289.

[30] Salza, P., Ferrucci, F., Sarro, F., 2016. Develop, deploy and execute parallel genetic algorithms in the cloud. In: GECCO Proceedings Companion, 2016. ACM, pp. 121–122.

[31] Sato, H., Aguirre, H. E., Tanaka, K., July 2010. Pareto partial dominance moea and hybrid archiving strategy included cdas in many-objective optimization. In: IEEE Congress on Evolutionary Computation. pp. 1–8.

[32] Sobeyko, O., Monch, L., 2017. Integrated process planning and scheduling for large-scale flexible job shops using metaheuristics. International Journal of Production Research 55 (2), 392–409.
URL https://doi.org/10.1080/00207543.2016.1182227

[33] Tao, F., Zhang, L., Venkatesh, V. C., Luo, Y., Cheng, Y., 2011. Cloud manufacturing: a computing and service-oriented manufacturing model. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 225 (10), 1969–1976.
URL https://doi.org/10.1177/0954405411405575

[34] Wan, J., Tang, S., Li, D., Imran, M., Zhang, C., Liu, C., Pang, Z., Jan 2019. Reconfigurable smart factory for drug packing in healthcare industry 4.0. IEEE Transactions on Industrial Informatics 15 (1), 507–516.

[35] Wang, S., Wan, J., Zhang, D., Li, D., Zhang, C., 2016. Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination. Computer Networks 101, 158 – 168, industrial Technologies and Applications for the Internet of Things.

[36] Zhang, L., Wong, T., 2013. Distributed genetic algorithm for integrated process planning and scheduling based on multi agent system. IFAC Proceedings Volumes 46 (9), 760 – 765, 7th IFAC Conference on Manufacturing Modelling, Management, and Control.
URL http://www.sciencedirect.com/science/article/pii/S1474667016343798

[37] Zhang, Q., Li, H., Dec 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Trans. on Evolutionary Computation 11 (6), 712–731.

[38] Zhao, S., Dziurzanski, P., Przewozniczek, M., Komarnicki, M., Indrusiak, L. S., 2019. Cloud-based dynamic distributed optimisation of integrated process planning and scheduling in smart factories. In: Proceedings of the Genetic and Evolutionary Computation Conference 2019. GECCO '19. ACM, New York, NY, USA, pp. 1381–1389.